Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Aplicación “pulsar para hablar” para dispositivos móviles con compatibilidad JME y Bluetooth.

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Jose Luis Puig Guerra

DIRECTOR: Juan Hernandez Serrano

FECHA: 17 de Julio de 2009

Titulo: Aplicación "pulsar para hablar" para dispositivos móviles con compatibilidad JME y bluetooth

Autor: Jose Luis Puig Guerra

Director: Juan Hernández Serrano

Fecha: 17 de Julio de 2009

Resumen

Actualmente, y cada vez más, la gente se comunica a través del móvil. Este dispositivo ha revolucionado nuestra forma de comunicarnos, ya sea con personas que estén cerca como con las que estén lejos. Dada la evolución de los teléfonos móviles, no es de extrañar que para hablar con personas cercanas se utilice el móvil en vez de una comunicación directa, ya que cada vez es menos necesario el contacto humano para tal efecto.

El objetivo para este proyecto es la creación de un sistema de comunicación cercano, multiplataforma, gratuito y sin la necesidad de ningún otro sistema externo para su funcionamiento. Con dicho sistema de comunicación no se pretende substituir las llamadas realizadas desde un móvil a través del sistema global de comunicaciones (GSM - *Groupe Special Mobile*), más bien su objetivo es dar soporte a las comunicaciones a corta distancia.

Se ha desarrollado una aplicación basada en tecnología inalámbrica, encaminada a un funcionamiento ad-hoc, compatible con la gran mayoría de teléfonos móviles actuales. Concretamente, el objetivo no es otro que la realización de llamadas de corto alcance sin ningún coste por parte del usuario, similar al método utilizado en los walkie-talkie, “pulsar para hablar”. A tal efecto, se reduce el número de dispositivos utilizados por el usuario, ya que se implementa en el propio terminal móvil.

La aplicación se desarrolla en Java, dado que es una tecnología soportada prácticamente por todos los teléfonos móviles. También existen otras, como C++ [1], pero su uso en los móviles es mucho menor.

La interfaz radio elegida para la comunicación es Bluetooth, ya que está implementada al mismo nivel que Java, y no las redes locales inalámbricas [2] (WLAN – *Wireless Local Area Network*) debido a que prácticamente no está disponible en estos dispositivos. También cabe mencionar que WLAN 802.11 tiene un rango mayor de cobertura, pero por el contrario requiere de un consumo mayor, acortando el funcionamiento de éstos.

Title: Aplicación "pulsar para hablar" para dispositivos móviles con compatibilidad JME y bluetooth

Author: Jose Luis Puig Guerra

Director: Juan Hernández Serrano

Date: July, 17th 2009

Overview

Nowadays, people use more than ever cell phones for expanding their communication needs. Those devices have revolutionized the way we interact with others, no matter whether they far or near us. Today, cell phones are in many cases the way of communicating and human contact becomes scarce.

The target of this TFC is to create a nearby communication system being cross-platform, free and without the need for any external system for its operation. It is not intended to replace the GSM cell phone calls but to support close-range communications.

We have developed an application based on wireless ad-hoc technology that it is compatible with most mobile phones today. Specifically, the goal is to provide a method for short-range calls at no cost, similar to the "push to talk" methods used e.g. in walkie-talkies.

The application is developed in JME because it is supported by almost all cell phones today. There are others potential language, such as C + + [1], but their use in mobile phones is much lower.

The interface chosen for the radio communication is Bluetooth because it is available in many mobile devices and has low power consumption.

ÍNDICE

TABLAS

FIGURAS

INTRODUCCIÓN

Conforme avanza la tecnología, ésta se ve reflejada, cada vez más, en los móviles, ya que es una herramienta indispensable en la sociedad actual. No es de extrañar, que cada nuevo modelo de móvil incorpore mas funciones adicionales y soporte mas complementos. Actualmente existen diversos servicios en los móviles que nos facilitan ciertas tareas, ya sea sacar dinero de un banco o comprar un producto; por ello se ha decidido realizar una aplicación para la comunicación a corto alcance, multiplataforma, libre y sin ningún coste para el usuario.

Este TFC presenta como objetivo una comunicación de audio entre dos o más personas mediante la tecnología inalámbrica bluetooth, también llamado 802.15.1, con encaminamiento ad-hoc. Con ello se pretende conectar un terminal con otro que no este a su alcance mediante un tercer terminal que haga de intermediario entre ambos, siempre y cuando este al alcance de los dos. Esta función permite establecer conexiones entre redes, tal y como ocurre con las redes scatternet.

Se ha elegido esta aplicación por diversas razones. La principal es la necesidad de comunicación entre usuarios cercanos sin la necesidad de depender de otros dispositivos. Con ello se obtiene una reducción del número de dispositivos que se portan cada día. La segunda razón es la comodidad de enviar grabaciones de voz en vez de mensajes de texto. Establecer una comunicación fluida mediante el uso de mensajes es muy difícil debido al tiempo que se tardan en escribir estos. Por el contrario, una comunicación con voz siempre es mas fluida y, sobretodo, mas clara y concisa. La ultima es puramente económica, ya que con dicha aplicación no requiere de un operador para realizar las comunicaciones, por lo que no se aplica una tarificación en los mensajes de audio enviados y/o recibidos. Cabe mencionar que cada persona suele tener como mínimo un terminal móvil, por lo que la posibilidad de penetración en la sociedad es muy elevada.

Debido a la gran influencia de los terminales móviles la aplicación ha sido diseñada para ser utilizada en la inmensa mayoría de móviles, ya que utiliza Java, concretamente la versión *Java 2 Micro Edition*. Actualmente Java es la tecnología más utilizada a la hora de usar aplicaciones, ya que proporciona fiabilidad, facilidad de uso, estandarización, y tal vez el factor más importante, multiplataforma. Además se han utilizado los perfiles y configuraciones mas antiguas para poder así aumentar la compatibilidad con los móviles. Se podría haber elegido C++, pero ello conllevaría limitar el número de dispositivos compatibles debido a su menor implementación el mercado actual. Éste también es uno de los factores a la hora de escoger bluetooth frente a WLAN. En este caso bluetooth es el más utilizado en los terminales frente a WLAN, además de tener un consumo mas reducido. Por el contrario, bluetooth tiene un alcance bastante limitado, aunque gracias a la posibilidad de formar redes scatternet, dicho alcance es ampliamente incrementado.

El documento se organiza como se detalla a continuación. El primer capitulo trata de introducir la tecnología utilizada, Bluetooth. Se detallan las características principales y las que son utilizadas en este proyecto, como por ejemplo las redes scatternet. Seguidamente, en el capitulo 2 se repasa el lenguaje Java, haciendo hincapié en la versión J2ME, que es la utilizada para realizar el proyecto. También se da una breve pincelada a la IDE utilizada, NetBeans, para poder compilar y simular. En el tercer capítulo se detalla la aplicación de envió de voz, mostrando su correcto funcionamiento con el simulador y en terminales reales. Finalmente el último capítulo se extraen las conclusiones junto con posibles líneas y un breve estudio medioambiental.

CAPÍTULO 1: BLUETOOTH Y REDES SCATTERNET

Bluetooth es una especificación industrial diseñada para redes inalámbricas de área personal (WPAN - *Wireless Personal Area Network*) con unos dispositivos de bajas prestaciones y bajo consumo, como pueden ser PDAs, ratones, mandos a distancia, etc.

Actualmente existen otras formas de comunicación inalámbrica en este tipo de dispositivos, como es el caso de los infrarrojos, pero su uso es cada vez menor, ya que proporciona un ancho de banda menor (115.2 kbps en su primera versión frente a los 3Mbps del Bluetooth 2.0) y peor cobertura, ya que su antena no irradia de forma omnidireccional dificultando la comunicación con el resto de dispositivos. Por estos motivos, Bluetooth es la tecnología ideal para las comunicaciones a corta distancia.

Bluetooth permite la conexión de dispositivos mediante redes ad-hoc e infraestructura, permitiendo descubrir dispositivos cercanos y establecer una comunicación con ellos de forma rápida y sencilla, dejando así atrás la necesidad de cables para las comunicaciones.

1.1. Historia

El nombre de esta tecnología Bluetooth proviene del siglo X, cuando el rey danés (940-981) Harald Blåtand conocido en su época como “Blatand”, “Bla” por su tez oscura y “tand” por su gran estatura, unifico los pueblos de Dinamarca, Suiza y Noruega.

El logotipo de Bluetooth (**Fig. 1.1**) representa la unión de las runas nórdicas H ᚼ(Hagall) y B ᛒ(Berkanan), haciendo referencia a las iniciales del rey danés Harald Blåtand. El color azul característico de esta tecnología se debe también al rey danés, ya que su apellido Blåtand fue traducido al inglés como Bluetooth cuya traducción es “dienteazul”.

Fig. 1.1 Logotipo de Bluetooth.

En honor a este rey, adoptaron el nombre de Bluetooth para su nueva tecnología ya que pretende unificar las comunicaciones entre diferentes periféricos, tal y como hizo Harald con las tribus de Dinamarca, Suiza y Noruega.

En 1994 la empresa sueca Sony Ericsson inició un estudio para investigar la viabilidad de una interfaz radio capaz de ser económica y competitiva en el mercado. El proyecto se llamó MC link y conforme avanzaron sus estudios otras empresas mostraron interés. Para asegurarse el posible éxito de esta tecnología, en 1998, se formó el grupo de trabajo de interés especial (SIG – *Special Interest Group*) siendo sus integrantes las cinco empresas más representativas de la época; Sony Ericson, IBM, Intel, Nokia y Toshiba.

El proyecto de SIG tuvo una gran acogida entre los otros fabricantes y a finales del mismo año ya eran mas de 1800 miembros. El objetivo era poder establecer un estándar para las comunicaciones vía radio a bajo coste, pudiendo competir con las comunicaciones por cable de aquel entonces.

No fue hasta 1999 cuando se publicó la primera especificación de Bluetooth. Durante los siguientes años, el grupo SIG tuvo un crecimiento muy rápido, siendo más de 9000 miembros en la actualidad. En 2002 el IEEE aprobó dicha especificación como 802.15.1, consolidando así la tecnología como un referente en las comunicaciones inalámbricas de red personal. Desde entonces se han especificado diferentes versiones de Bluetooth, cada una con mayores prestaciones, aunque siempre con especial interés en un mínimo consumo y mayor facilidad de uso. Según la versión, su velocidad varía sustancialmente (**Tabla 1.1**).

Tabla 1.1 Velocidades según las versiones.

Versión	Velocidad
1.1	537 Kbps
1.2	1 Mbps
2.0	3 Mbps
3 (no especificado aún)	480 Mbps

1.2. Especificaciones Bluetooth

La especificación Bluetooth es muy extensa, al no ser el tema a tratar, se explican las partes brevemente (**Fig. 1.2**).

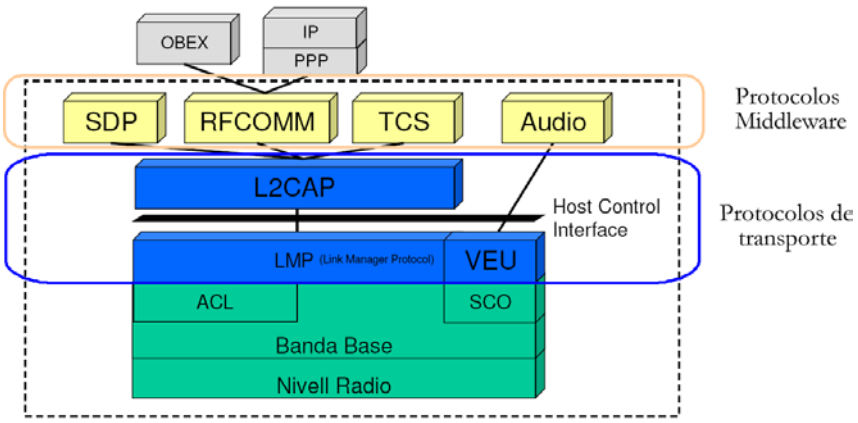

Fig. 1.2 Arquitectura Bluetooth.

1.2.1. Radio

Esta tecnología opera en la banda de 2,4 Ghz., lo que significa que trabaja en la banda libre, por lo que su uso no es comercial.

No solo Bluetooth trabaja en esta banda, si no que WLAN también. Debido a este problema de interferencia entre las dos tecnologías, Bluetooth incorpora el sistema de espectro ensanchado por salto de frecuencia (FHSS – *Frequency Hopping Spread Spectrum*) que permite los saltos entre frecuencias en tan solo 625µs. Esta solución también ayuda a que la intensidad de la señal sea más constante.

La modulación adoptada es la de desplazamiento de frecuencia gausiana (GFSK – *Gaussian Frequency Shift Keying*) ya que conjuntamente con la división duplex en tiempo (TDD – *Time Division Duplex*) permiten un espectro más estrecho y una velocidad mayor.

No todos los dispositivos que implementan Bluetooth tienen las mismas capacidades, por lo que se definen tres clases (**Tabla 1.2**).

Tabla 1.2 Potencias según las clases de terminales.

Clase	Potencia máxima	Potencia minima	Control de potencia
1	100mW	1mW	Requerido
2	2.5mW	0.25mW	Opcional
3	1mW	No requerido	Opcional

1.2.2. Enlaces físicos

Según el tipo de servicio que se requiera, se ofrece un tipo de enlace de los dos disponibles:

- Síncrono: Este tipo de enlace, también llamado SCO (*Synchronous Connection Oriented*) se utiliza cuando se quiere transportar un tráfico prioritario, como es el caso de la voz. Se trabaja con reserva de *slot* y no admite la recuperación de paquetes.

- Asíncrono: El enlace asíncrono no orientado a conexión (ACL - *Asynchronous Connection-less*) es el enlace más utilizado ya que transporta casi todo el tráfico. Permite retransmisiones y utiliza el esquema de *polling* para conceder recursos.

Con estos dos tipos de enlaces puede diferenciar el tráfico, ofreciendo una cierta calidad de servicio (QoS – *Quality of Service*) necesaria en ciertos servicios, como es el caso de la voz.

1.2.3. LMP

El controlador de estado del protocolo (LMP - *Link Manager Protocol*) entra dentro del apartado de control de acceso al medio (MAC - Media Access Control) del modelo de interconexión de sistemas abiertos (OSI – *Open System Interconnection*) y su principal función es la de gestionar el enlace establecido entre dos o más dispositivos. También es el encargado de gestionar la seguridad del enlace, ofreciendo autenticación y encriptación.

Con las unidades de datos del protocolo (PDU – *Protocol Data Unit*) de esta capa se controla el sincronismo y se negocian las capacidades según sea de una clase u otra.

1.2.4. HCI

La interfaz de control de maquina (HCI - *Host Control Interface*) permite un acceso universal mediante comandos, para conocer o modificar el estado del enlace. Este protocolo controla el buffer del canal ACL, pudiendo desactivarlo si es necesario.

Tal es la importancia del ahorro en el consumo del dispositivo, que parte de esta capa puede ser implementada en un *Host*.

Como es lógico, esta capa define sus propias PDU's, aunque no serán tratadas en este proyecto, al no ser el tema principal.

1.2.5. L2CAP

El protocolo de adaptación y control de enlace a nivel lógico (L2CAP - *Local Link Control and Adaptation Protocol*) multiplexa, fragmenta y adapta los datos provenientes de las capas adyacentes (LMP y los diferentes protocolos middleware). Es decir, adapta las PDU's al siguiente nivel. También realiza la función de la gestión de QoS, diferenciando el canal necesario para ofrecer cada servicio. Cabe destacar que solo se implementa en el canal ACL, ya que el SCO es gestionado por el protocolo *middleware* directamente.

1.2.6. Protocolos Middleware

Estos protocolos son los que nos permiten utilizar Bluetooth, ya que hacen posible el funcionamiento de los dispositivos. Hay cuatro protocolos definidos en el estándar, aunque existen otros.

Tabla 1.3 Protocolos middleware.

Protocolo	Función
RFCOMM	Emula el puerto RS-232 con un máximo de 60 puertos concurrentemente. Posee un control de flujo sobre el canal pero no sobre los errores.
TCS	Establece y termina las llamadas. Gestiona y transporta las datos de las llamadas.
SDP	Permite conocer los servicios disponibles en los dispositivos.
OBEX	Define el intercambio de objetos entre terminales y implementa un sistema de sincronización automática, con tan solo acercarse.

1.2.7. Estados de la conexión

Estos estados pertenecen a la capa LMP ya que es la que gestiona el enlace entre los dispositivos. Reflejan las diferentes etapas que se realizan para establecer una conexión.

Tabla 1.4 Estados de la conexión.

Estado	Función
Standby	No forma parte de ninguna red.
Inquiry	Busca dispositivos a su alcance.
Page	Calcula el FHSS y espera a que estén coordinados.
Transmision	Forma parte de una red.
Connected	Puede enviar y recibir datos.

1.2.8. Estados del terminal

Uno de los principales objetivos de Bluetooth es el ahorro en el consumo de batería. Debido a esta importante reseña, hay definidos tres estados.

Tabla 1.5 Estados del terminal.

Estado	Función
Sniff	Escucha el medio en determinados intervalos.
Hold	Si no hay trafico, se puede cerrar el canal ACL pero no el SCO.
Park	Se despierta en determinados intervalos para no perder la sincronización.

Estos estados son capaces de ahorrar un porcentaje muy representativo del consumo del terminal. Para tener una idea de cuan grande es el ahorro, se muestra la (**Tabla 1.6**) realizando una comparación con un dispositivo que implemente 802.11b (WLAN)

Tabla 1.6 Comparación de consumo Bluetooth-WLAN.

Estado	802.15.1 [mA]	802.11b [mA]
Sniff	53	300
Park	4	200
Hold	0,047	20

1.3. Perfiles

Los perfiles son un conjunto de mensajes y procedimientos para una determinada situación de uso del dispositivo.

Este sistema ofrece ciertas ventajas, como la de no tener que implementar toda la pila de protocolos en un dispositivo. Asimismo garantiza la interoperabilidad entre terminales.

Hay multitud de perfiles implementados en la especificación, éstos son resumidos brevemente en la (**Tabla 1.7**).

Tabla 1.7 Perfiles Bluetooth.

Perfil	Descripción
A2DP	Define como transmitir un stream de audio.
AVRCP	Ofrece un interfaz para el control de aparatos de música, televisores, etc.
BIP	Controla el envío, conversión y partición de imágenes.
BPP	Permite el envío de archivos para su posterior impresión.
CIP	Concede el acceso a ISDN.
CTP	Ofrece un servicio para teléfonos inalámbricos.
DID	Permite identificar un dispositivo, su utilidad es parecida a la PlugAndPlay.
DUN	Define un acceso a red mediante dial-up.
FAX	Permite el envío de fax.
GAVDP	Define la forma de trabajar con audio/video.
GAP	Se encarga de descubrir y establecer la comunicación.
GOEP	Permite el intercambio de objetos entre terminales.
HCRP	Ofrece una comunicación inalámbrica entre dispositivo e impresora.
HFP	Controla la función de “manos libres”.
HID	Da soporte a dispositivos tales como: ratones, teclados y joysticks.
HSP	Permite el uso de auriculares.
ICP	Ofrece un servicio equivalente al “walkie-talkie”.
OPP	Define el envío de objetos con imágenes.
PAN	Permite el uso del protocolo de encapsulación Bluetooth.
PBAP	Controla el acceso a la agenda de teléfonos.
SPP	Define el puerto serie.
SDAP	Descubre los servicios disponibles.
SAP, SIM	Ofrecen un acceso a la SIM
SYNCH	Controla la sincronización.
VDP	Habilita el transporte de video en stream.
WAPB	Permite el transporte de WAP sobre PPP.

1.4. Redes piconet

El termino *piconet* se refiere cuando se conectan dos dispositivos mediante Bluetooth compartiendo un único canal físico, creando así una red.

Esta configuración define un dispositivo como maestro, dejando al resto como esclavos. Este maestro controla la comunicación y como máximo puede llegar a tener siete esclavos simultáneamente.

En el caso que requiera de más usuarios conectados, puede pedir a un esclavo que pase al estado *park*, dejando su sitio libre para otro dispositivo.

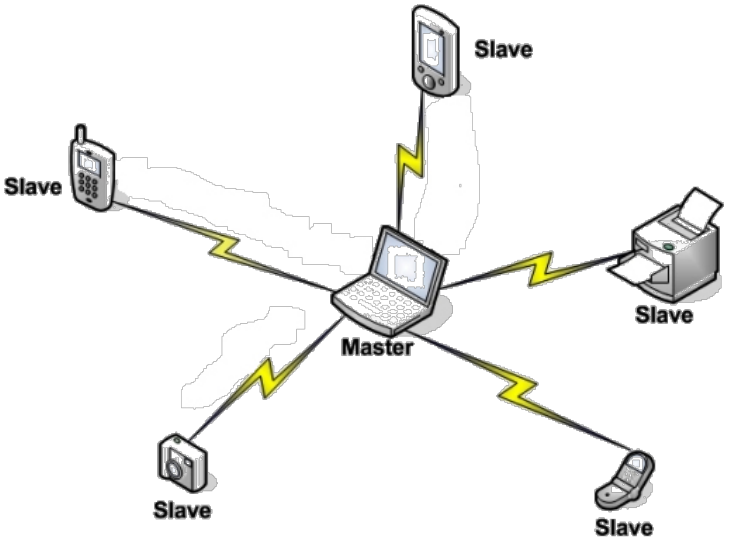

Fig. 1.3 Piconet.

1.5. Redes scatternet

En el caso de requerir una red mayor, Bluetooth define la estructura llamada *scatternet*. Esta red esta formada por dos o más *piconets*, por lo que el funcionamiento es muy similar a ésta. Un dispositivo que esté al alcance de las dos o más *piconets* hace de puente participando un cierto tiempo en cada red. Estos puentes pueden transportar información de una red a otra, de aquí su nombre “puente”.

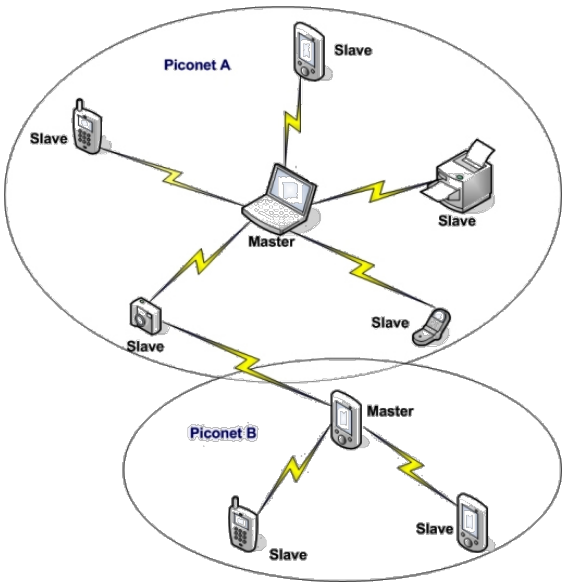

Fig. 1.4 Scatternet.

1.6. Seguridad

Bluetooth ofrece seguridad en sus comunicaciones, ya que a través de sus enlaces se transmiten datos privados, como por ejemplo realizar una compra a través de este sistema.

Por tal razón, contiene tres distintos modos de seguridad, impuestos por cada fabricante.

- No seguro.
- Seguridad a nivel de enlace.
- Seguridad a nivel de servicio.

Aparte de estos modos, también cataloga tres niveles de seguridad en los servicios.

- Servicios abiertos.
- Servicios con autenticación requerida.
- Servicios con autenticación y autorización necesaria.

Adicionalmente, también es posible vincular un dispositivo con un cierto nivel de seguridad, permitiendo acceso (de confianza) o negándoselo (de NO confianza).

Incluso con estas definiciones de seguridad, Bluetooth no es un medio 100% seguro, ya que utiliza el medio radio para comunicarse. A través del tiempo se han descubierto vulnerabilidades en el sistema, siendo las más conocidas Bluehacking [3] y Bluesnarfing [4].

Bluehacking es la menos peligrosa de las dos, ya que tan sólo genera *spam* mediante mensajes de texto. Por el contrario Bluesnarfing es más peligroso debido a que afecta a la funcionalidad de los terminales, ya sea clonando el dispositivo o robando datos.

CAPÍTULO 2: HERRAMIENTAS DE TRABAJO

Este capítulo trata de dar una ligera impresión sobre las herramientas utilizadas para realizar el proyecto. Consta de una breve introducción al primer lenguaje orientado a objetos, Java. Acto seguido se explica Java 2 edición micro (J2ME – *Java 2 Micro Edition*), que es la versión utilizada en este caso. Finalmente concluye con el entorno de desarrollo integrado (IDE – *Integrated Development Enviorment*) NetBeans, indispensable para la creación de la aplicación.

2.1. Java

Java es un lenguaje de alto nivel, con multitud de características como, ser: sencillo, robusto, distribuido, multitarea, dinámico y orientado a objetos; es decir, un lenguaje de programación completo.

Java inició sus pasos con aplicaciones interactivas integradas en Webs, llamados applets, y con aplicaciones independientes, conocidas como aplicaciones *stanalone*. Hoy en día se utiliza en servicios de transferencia de de hipertexto (HTTP – *HyperText Transfer Protocol*), servidores y incluso bases de datos java [5] (JDBC – *Java DataBase Connection*), por tal motivo no es de extrañar que este lenguaje no lleve dos décadas y ya es uno de los más utilizados en muchos entornos de programación.

2.1.1. Historia

Este lenguaje fue diseñado, a finales de la década de los 90 por un grupo de trabajo de Sun Microsystems, denominado *Green Team*. Al ser desarrollado en los 90, antes de la explosión del “World Wide Web” [6], fue diseñado para dispositivos electrónicos como calculadoras, microondas, televisión etc.

Tras año y medio de trabajo, el *Green Team* presentó un lenguaje de programación al que llamaron *Oak*. Éste era un sistema que se podía implementar en cualquier plataforma, ya que existían multitud de modelos en el mercado. Entre las muchas características que se requirieron en el proyecto, la sencillez de programación y la robustez fueron claves para su desarrollo.

Mas tarde, y por cuestiones de propiedad intelectual, adopto el nombre de Java dado que el anterior nombre ya era un lenguaje de programación. Al cambiar el nombre, adoptaron la mascota “Duke” [7] (**Fig. 1.5**) para dar mas empuje a la tecnología.

Fig. 2.1 Mascota de Java.

Debido a la falta de dispositivos que implementasen esta tecnología, el proyecto Java quedó estancado y no fue hasta mediados de 1995, tras la llegada del World Wide Web, que se rescató para ser utilizado en el navegador Netscape Navigator.

Actualmente, más de 4.500 millones de dispositivos, entre los que se encuentran equipos personales, teléfonos móviles, tarjetas inteligentes y otros, implementan Java, convirtiéndola en un referente mundial.

2.1.2. Estándares Java 2

Debido a la gran variedad de dispositivos que implementan Java y dado que las necesidades han aumentado, se han desarrollado diferentes soluciones específicas para cada ámbito tecnológico.

De estas necesidades, surgieron las ediciones Java 2 edición estándar (J2SE – *Java 2 Standard Edition*), destinado a applets y aplicaciones independientes, Java 2 edición empresa (J2EE – *Java 2 Enterprise Edition*) para aplicaciones empresariales y Java 2 edición micro (J2ME – *Java 2 Micro Edition*) especificado para dispositivos limitados en memoria y capacidad de proceso, tales como teléfonos móviles (**Fig. 1.6**).

Fig. 2.2 Estándares de Java.

Estas versiones no son independientes entre sí, tan sólo son ediciones con funciones dedicadas para cada uso.

2.1.2.1 Java 2 Estandard Edition

Esta versión es la base de Java, necesaria para la edición J2EE. Implementa el compilador, herramientas, módulos de ejecución y la interfaz de programación de aplicaciones (API – *Application Programming Interface*). Opera en cualquier sistema operativo ejecutado sobre la maquina virtual java.

Las herramientas ofrecidas por esta versión permiten el desarrollo de aplicaciones y applets. La versión mas reciente es la 1.6, aunque se espera otra en poco tiempo.

2.1.2.2 Java 2 Enterprise Edition

También conocido como J2EE, se basa en la edición Standard Edition y tiene como finalidad ser implementado en entornos empresariales. Es capaz de trabajar de forma distribuida y escalable sobre componentes modulares gobernados por un servidor de aplicaciones.

Esta especificación incorpora distintas API's, tales como: JDBC (base de datos), RMI (usos remotos), e-mail, servicios web, XML…etc.

La ultima versión, 6, está disponible para la descargar en la pagina de Java, como los demás estándares.

2.1.2.3 Java 2 Micro Edition

Es la especificación utilizada en este proyecto, sus características se verán más detalladas en el siguiente capitulo.

Sus prestaciones son enfocadas a los dispositivos con una capacidad computacional y de memoria reducida ya que es un entorno altamente optimizado para determinados usos.

Cabe resaltar que esta edición es una versión "reducida" de J2EE y no incorpora algunos tipos de datos y ciertas operaciones matemáticas complicadas.

2.1.3. Java Virtual Machine

Para poder ejecutar los códigos realizados en cualquiera de las especificaciones de Java, se requiere una maquina virtual (JVM – *Java Virtual Machine*). Su función es interpretar y ejecutar los códigos en binarios creados en Java mediante un compilador, llamados *bytecodes*. Estos códigos binarios son interpretados y adecuados a cada tipo de software y hardware.

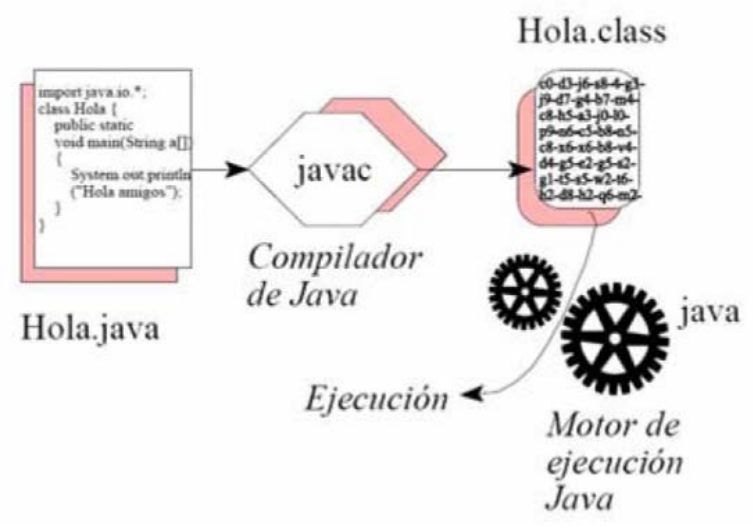

Fig. 2.3 Compilación y ejecución en Java.

La principal característica de esta maquina virtual, es su posibilidad de ejecución en cualquier entorno. Esto le ha valido a Java el apelativo de "*Write once, run anywhere*", traducido al castellano como "Escríbalo una vez, ejecútelo en cualquier parte".

2.2. Java 2 Micro Edition

La creación de esta edición de Java supuso la necesidad de alguna máquina virtual que se ajustara a las capacidades de los terminales. A raíz de esta necesidad se especificaron dos maquinas virtuales, máquina virtual *Kilobyte* (KVM - Kilobyte Virtual Machine) y maquina virtual compacta (CVM - *Compact Virtual Machine*), correspondiendo una diferente configuración para cada una.

2.2.1. Maquinas virtuales de J2ME

Al abarcar J2ME tantos tipos de dispositivos diferentes, existen dos maquinas virtuales con una configuración determinada. Siendo estas CLDC (*Connected Limited Device Configuration* – Configuración de dispositivos limitados) y CDC (*Connected Device Configuration* – Configuración de dispositivos).

La maquina virtual *Kilobyte* es utilizada en la configuración CLDC mientras que la maquina compact es en CDC.

2.2.1.1 Kilobyte Virtual Machine

Esta maquina virtual requiere de una capacidad y memoria muy reducida, haciendo su nombre referencia al espacio requerido para ésta. Está escrita en

C y apenas llega a las 24.000 líneas de código, razón por la cual ocupa tan poco espacio.

Sus principales características son:

- Utilizada en configuraciones CLDC.
- Altamente portable.
- 1 Kilobyte de espacio requerido.
- Suprime algunas características, pero no sacrifica los requisitos necesarios.
- Consumo de 100 KB de memoria en aplicaciones típicas.

2.2.1.2 Compact Virtual Machine

Esta máquina está diseñada para dispositivos electrónicos y contiene las mismas características que su homónima en J2SE, siendo éstas:

- Orientado a arquitecturas de 32 bits.
- Mínimo 2 MB de memoria RAM.
- Ejecuciones de clases Java fuera de la memoria ROM.
- Recolector de basuras modularizado.

2.2.2. Configuraciones

Las configuraciones son un conjunto de API's que hacen posible el funcionamiento de aplicaciones en un grupo de dispositivos determinado.

Estas API's describen las propiedades básicas de los terminales, especificando las características del lenguaje Java y de la propia maquina virtual, así como las bibliotecas básicas.

A tal efecto, y como se ha comentado anteriormente, existen dos configuraciones definidas; CLDC y CDC.

2.2.2.1 CLDC

Los tipos de dispositivos que utilizan esta configuración son teléfonos móviles, PDA, buscapersonas…etc. Suelen soportar conexiones, en su mayoría inalámbricas e intermitentes, con un consumo y velocidad reducidos, además de una seguridad tipo sandbox.

Para su uso, el terminal debe de disponer:

- Procesador de 16 bits, con una velocidad de 25 Mhz.
- Como mínimo 160 Kb de memoria.
- 128 Kb de memoria no volátil.
- 32 Kb de memoria volátil para tiempo de ejecución.

2.2.2.2 CDC

Decodificadores de televisión, electrodomésticos, sistemas de navegación…etc. son los dispositivos que utilizan esta configuración. Para su funcionamiento se requiere:

- Procesador de 32 bits.
- 2 Mb o más de memoria, incluyendo RAM y ROM.

Como se puede apreciar, los requisitos son superiores a los de CLDC, de aquí que esta configuración tan sólo tenga limitaciones gráficas y de memoria.

2.2.3. Perfiles

Los perfiles son un conjunto de clases de Java que implementan una configuración para un grupo determinado de dispositivos.

Los perfiles controlan los aspectos:

- Ciclo de vida de la aplicación.
- Interfaz de usuario.
- Conexiones de red.
- Ciertos aspectos de seguridad.
- Permite la portabilidad de aplicaciones J2ME entre dispositivos.

Dependiendo del tipo de configuración utilizada, CDC o CLDC, hay disponibles ciertos perfiles:

Perfiles disponibles con CLDC:

- Perfil de dispositivo para información en mobiles (MIDP - *Mobile Information Device Profile*).
- Perfil PDA.

Perfiles disponibles con CDC:

- Perfil base (FP - *Foundation Profile*).
- Perfil personal (PP- *Personal Profile*).
- Perfil de invocación de método remoto (RMI – *Remote Method Invocation*).

2.2.3.1 Foundation profile

Este perfil define las API’s necesarias para dispositivos que requieren interfaz gráfica, como por ejemplo decodificadores de televisión. Incluye gran parte de librerías de J2SE, pero deja aparte las necesarias para la interfaz de usuario

(GUI – *Guide User Interface*). Si se desease una GUI, seria necesario un perfil adicional.

2.2.3.2 Personal Profile

Proporciona un entorno completo para soporte gráfico, además de capacidades Web y *applets*.

Para poder utilizar este perfil, es necesario implementar también el perfil "*Foundation Profile*", ya que es la base para este perfil.

2.2.3.3 RMI Profile

Soporta el subconjunto de API's de J2SE, aunque algunas funciones se han descartado debido a su complejidad computacional y uso de memoria.

Este perfil, como el PP, necesita el "Foundation Profile" para poder ser utilizado.

2.2.3.4 Mobile Information Device Profile

Es el perfil más utilizado, ya que la mayoría de aplicaciones se basan en él. Permite la ejecución de aplicaciones Java sobre un dispositivo.

Hasta el momento se han publicado dos versiones MIDP 1.0 y 2.0 teniendo esta última, mejoras significativas respecto a la interfaz gráfica, conectividad y seguridad.

2.2.3.4 PDA Profile

Como su nombre indica, está diseñado para PDAs de gama baja, tipo Palm, con algún tipo de puntero (ratón o lápiz).

Este perfil todavía esta en fase de desarrollo, por lo que sus características no son bien conocidas.

2.3. NetBeans

NetBeans es una herramienta extensible y modular. Esta escrita en Java, y por eso deja de ser portable a cualquier sistema operativo. Cabe resaltar que esta herramienta tiene licencia CDCL [8] (*Common Development and Distribution Licence* – Licencia común de desarrollo y distribución) por lo que es gratuita.

Esta IDE trabaja con un conjunto de componentes software llamados "módulos". Gracias a esta filosofía de trabajo, NetBeans ofrece un sin fin de posibilidades.

Los módulos son clases de java capaces de interactuar con la API de Netbeans, siendo reconocidos gracias a un archivo especial llamado “manifiesto”.

Gracias a estos componentes software, NetBeans es un referente a la hora de programar aplicaciones, ya que soporta un gran número de lenguajes (**Tabla 2.1**).

Tabla 2.1 Módulos de NetBeans.

Modulo	Descripción
Enterprise pack	Ofrece editores de esquemas XML, WSDL y BPEL.
Profiler	Para optimizar aplicaciones Java.
Visual Web	Provee una forma senzilla de crear aplicaciones web con AJAX y JSF.
Ruby	Permite crear aplicaciones en ruby.
PHP	Permite crear aplicaciones con PHP, con un debugger integrado.
Phyton	Compatible con el lenguaje phyton.
C/C++	Permite diseñar proyectos con estos lenguajes.
Movility pack	Se especifica para la creación de aplicaciones en teléfonos móviles, permitiendo escribir, depurar y probar.

Debido a su estructura modular y a su distribución gratuita, NetBeans es la herramienta de de programación elegida por la mayoría de usuarios, tanto profesionales como aficionados.

CAPÍTULO 3: DESARROLLO DE LA APLICACIÓN

A continuación se explica la creación de un proyecto desde el principio mediante la IDE comentada anteriormente. Tras esta explicación, se detallan las fases de creación del proyecto: interfaz grafico, comunicación cliente – servidor e implementación del multi-salto.

Finalmente, se muestra un resumen de las clases creadas en el proyecto, especificando sus funciones y características.

3.1. Creación de proyectos con NetBeans 6.7

Como se ha comentado anteriormente, NetBeans es una aplicación gratuita y está disponible en http://www.netbeans.org/ para su descarga. También ofrece la posibilidad de pedir el DVD con todas las aplicaciones de dicha IDE de forma gratuita.

En el caso de elegir la opción de descarga, Sun ofrece soporte para los sistemas operativos más comunes; Windows, Linux, Mac OS X y Solaris además de una gran variedad de idiomas.

Gracias a esta IDE, la creación de proyectos se realiza de forma rápida y sencilla, tan sólo hay que seguir unos pocos pasos:

- File → New Project
- Choose Project → Java ME → Mobile aplication → Next

Una vez elegido, se ofrecen dos opciones muy interesantes: “Set As Main Project” y “Create Hello MIDlet”. Esta última opción es utilizada como toma de contacto con J2ME y NetBeans, ya que se trata de un ejemplo básico de midlet, pudiendo ser probado en el propio simulador o en un terminal que implemente Java. La primera opción, por el contrario, no ofrece ninguna ventaja a nivel de programación, más bien simplifica el acceso a la herramienta de compilación.

- Name and Location → Next/Finish

En este apartado se ofrece la posibilidad de elegir el directorio donde se va a crear y el nombre del mismo.

- Default Platform Selection → Emulator Platform: SDK 3.0 → Device: DefaultColorPhone → Next/Finish

En esta parte se permite escoger entre diferentes configuraciones; CLDC 1.0 y CLDC 1.1, junto con las diferentes versiones de los perfiles MIDP; 1.0, 2.0 o 2.1.

Para este proyecto se ha utilizado MIDP-1.0 y CLDC- 1.0 ya que al ser las versiones más antiguas, también son las más implementadas en los móviles, obteniendo así una mayor compatibilidad.

En el siguiente menú, existe la posibilidad de agregar plantillas (*templates*), aunque en este proyecto no se han utilizado.

Por último, el entorno de trabajo resultante de la creación del proyecto (**Fig. 3.1**)

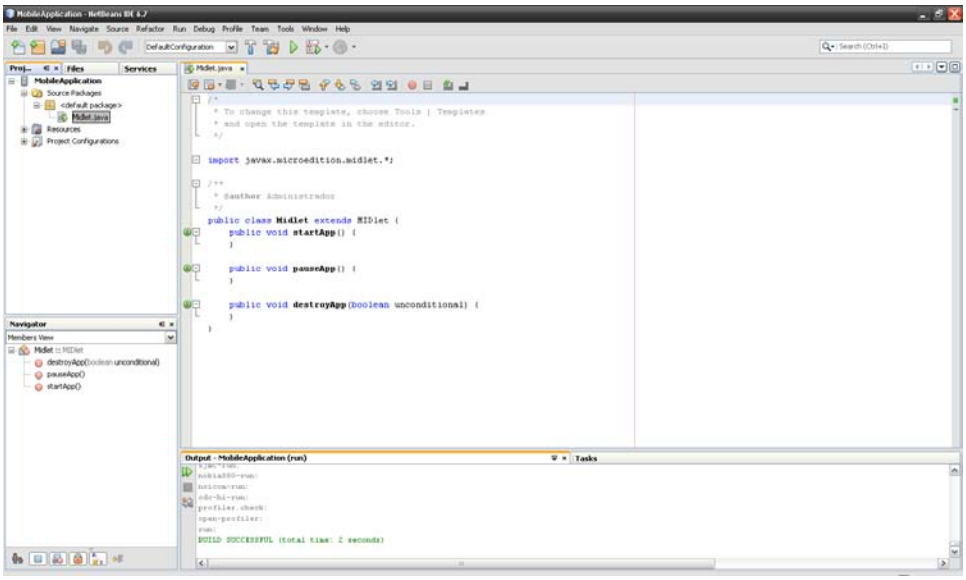

Fig. 3.1 Entorno de trabajo con NetBeans 6.7.

Como se observa, la pantalla está dividida en cuatro partes, siendo la superior izquierda la reservada para listar los posibles proyectos que hay en la carpeta de trabajo o *workspace*. En el mismo lado, pero en la zona inferior, se sitúa la subpantalla que muestra las clases, métodos y componentes que forman el proyecto. En el otro extremo, se encuentra la subventana dedicada a informar sobre el registro de compilación, ejecución y salida del programa. Finalmente, la ventana más grande es donde se introduce código para la aplicación pertinente.

Siguiendo su filosofía, NetBeans, permite editar de sitio las subventanas, añadir iconos, utilidades…

3.2. Fases de desarrollo

Para llevar a cabo el desarrollo de la aplicación, éste ha sido dividido en tres partes; interfaz gráfica, conexión del cliente con el servidor y la posibilidad de trabajar como las redes scatternet, es decir, establecer una conexión con un dispositivo que no está a su alcance directo pero sí al de otro cercano.

Tanto el cliente, servidor, interfaz gráfica y “salto de dispositivo” se han agrupado en la clase “`Midlet_audio`” convirtiéndola en la clase principal. Dentro de ésta, se encuentran diferentes clases, cada una con un propósito, detallando cada una en este capítulo.

3.2.1. Entorno gráfico

El entorno gráfico se ha desarrollado según las necesidades de la aplicación y las posibilidades de la IDE y J2ME. El diseño de éste ha sido enfocado a un usuario básico, para un fácil uso, dejando a un lado una interfaz complicada y farragosa.

Como cabe esperar, la interfaz evoluciona según requiera la aplicación, por lo que ha sufrido constantes cambios hasta llegar al resultado final. Posiblemente es la parte que menos trabajo pueda dar, si se compara con la parte de las conexiones y multi-salto.

2.1.1.1. Form

El formulario, `Form`, es un componente que actúa como contenedor de un número ilimitado de objetos. Estos pueden ser de cualquier tipo, pero siempre se ha de tener en cuenta que un objeto no puede estar ligado a dos formularios a la vez.

Para entender mejor este componente, el `Form` viene a ser la pantalla de un dispositivo móvil, donde se puede escribir y mostrar texto, imágenes, listas, botones… gracias al método `append`.

Para poder cambiar de formulario, se necesita el objeto comando, `Command`, ya que permite ejecutar un método al pulsar una tecla concreta en el móvil. Esta tecla no es “escuchada” si no se implementa el método `CommandListener`, que permite saber que `Command` ha sido pulsado el usuario.

Seguidamente, en la figura (**Fig. 3.2**), se expone un breve ejemplo sobre el uso de los `Form` en el propio simulador de la IDE, donde se pueden ver diferentes mensajes para informar al usuario.

Fig. 3.2 Ejemplo de uso del objeto `Form`.

2.1.1.2. ChoiceGroup

Esta clase permite mostrar una lista de elementos seleccionables dentro de un `Form`, pueden ser de tipo múltiple (se puede seleccionar más de un objeto) o exclusiva (tan sólo se puede seccionar uno). La forma de añadir objetos a la lista es casi idéntica a la de los formularios, ya que utiliza el método del mismo nombre, `append`, pero con diferentes argumentos.

Dispone de un método de escucha llamado `itemStateChanged` que permite realizar acciones si se selecciona un objeto de la lista, muy parecido al `CommandListener` de la clase `Form`.

Posteriormente se muestra un ejemplo de utilización de esta clase, donde se muestran los terminales encontrados en el formato explicado (**Fig. 3.3**).

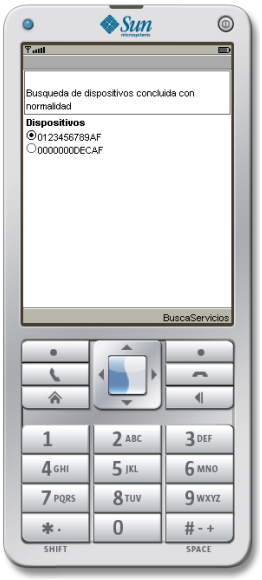

Fig. 3.3 Ejemplo de utilización de `ChoiceGroup`.

2.1.1.3. Command

Tal y como se ha explicado anteriormente, esta clase proporciona la comunicación necesaria entre el usuario y el programa, ya que por el momento, es la forma más sencilla para interactuar con el usuario. Se puede pensar en ello como un botón en Widows/Linux, por establecer una analogía.

Para su utilización se necesitan tres parámetros básicos: 1) nombre del objeto `Command` que se quiere mostrar en el `Form`, y que por tanto vea el usuario; 2) el tipo de objeto `Command`, hay unos pocos para elegir y dependiendo del tipo se muestran en un determinado sitio en el `Form`; y 3) la prioridad con la cual aparece en el comando en pantalla. Normalmente este último parámetro no es de mucha importancia en este tipo de aplicaciones debido a la rapidez con que estos aparecen.

La clase `CommandAction` es la encargada de “escuchar” el terminal en busca de acciones realizadas por el usuario, cuandos e ha llamado previamente el método `CommandListener` de la clase `Form`.

Seguidamente en la figura (**Fig. 3.4**) se muestra un formulario que contiene dos comandos: *rec* y *play*.

Fig. 3.4 Ejemplo de utilización de `Command`.

3.2.2. Comunicación cliente - servidor

Una vez concretado el apartado gráfico, se pasa a explicar el funcionamiento de la comunicación entre dos o más usuarios. Para esta comunicación, y en general todas, se requiere de un servidor que acepte las peticiones de conexión junto con un cliente, que se encarga de buscar y realizar las peticiones al servidor.

2.1.1.4. Comunicación

Debido a que las comunicaciones con un móvil no se limitan solamente a Bluetooth, J2ME implementa una serie de métodos para implementar las conexiones con los paquetes `javax.microedition.io` y `java.io`. Éstas permiten la conexión mediante sockets, datagramas, http… entre otras.

Ya que el proyecto se basa en comunicaciones 802.15.1, se tiene que utilizar la API `javax.bluetooth` para la comunicación. Este paquete de clases define dos nuevos tipos de conexión a los anteriores; SPP y L2CAP, comentados anteriormente.

Para poder comunicarse, se necesita crear el objeto de la comunicación mediante el método `open` de la clase `Connector`. Cuando se crea necesita diferentes parámetros, como el tipo de protocolo y dirección, además de varios opcionales.

Una vez se establece la conexión, éste retorna un objeto de la clase `StreamConnectionNotifier` que permite aceptar diferentes clientes mediante el método `acceptAndOpen`. Al llamar a este método, se retorna un

objeto de tipo `StreamConnection` que representa al cliente que está conectado en ese momento.

Con este tipo de objeto, se pueden establecer los flujos de entrada y salida, parte indispensable para intercambiar datos. Estos flujos, y más concretamente los utilizados en el proyecto, son los encargados de transportar datos en los dos sentidos de la comunicación. Se crean gracias a los métodos `openDataInputStream` y `openDataOutputStream` del objeto que representa al cliente. Tal y como su propio nombre indica, el primer método define el flujo de entrada, para recibir datos, mientras que el segundo es utilizado para enviar información.

Se han elegido estos flujos de entrada y salida debido a que son capaces de enviar cadenas de datos (*streams*), necesarios para transportar audio, además de diferenciar el tipo de información que contienen. En este proyecto se diferencian tres tipos de datos enviados a través de estos flujos, siendo; enteros (*int*), cadenas de texto (*string*) y arrays de bytes (*byte []*).

Para poder recibir correctamente cada tipo de información, se utilizan los métodos `readUTF`, para strings, `readInt`, en el caso de enteros y `readFully` para las arrays de bytes. En el caso contrario, para enviar información, se utilizan los mismos métodos pero en escritura; `writeUTF`, `writeInt` y `write`, respectivamente.

Cabe resaltar que este último método lee el número de bytes indicados por parámetro, por lo que si se indica leer 100 bytes, bloqueará el sistema hasta que los haya recibido todos.

2.1.1.5. Servidor SPP

Como ha sido comentado anteriormente, la complejidad de esta clase es mucho menor que la del cliente, debido a que este tan sólo acepta conexiones, en cambio el cliente tiene que buscar el dispositivo y el servicio.

Acto seguido se comentan las principales características del servidor creado, resaltando aquellas relacionadas con los servicios ofrecidos.

Para que un dispositivo pueda conectarse a un servidor, primero tiene que buscar el servicio que éste ofrece, para ello se utilizan una serie de atributos cuando se crea el servicio. Debido a esta necesidad, se crean los *UUID* [9] (*Universal Unique Identifier* – Identificador único universal), que no son más que un numero de 16 bits. Cada tipo de servicio tiene pre-asignado un número de estos en los estándares, aunque también pueden ser propios.

Este proyecto se basa en comunicaciones por puerto serie, definidos en el estándar, por lo que el UUID asignado es el 0x0003. El dato tiene que ser conocido por el cliente, como se verá más adelante.

Para poder trabajar con Bluetooth, es necesario obtener la configuración del dispositivo, ya que sin ella no es posible ningún tipo de conexión de este tipo. El objeto `LocalDevice` permite esta operación, como también la posibilidad de poner el dispositivo visible para posibles búsquedas en el medio físico con el método `setDiscoverable`.

Mediante la clase `Connector` se crea el objeto referente a la conexión, especificando el protocolo y la dirección física del móvil. En este proyecto la clase conector recibe los parámetros *btspp://localhost;name="Server"*.

El nombre, o *name*, permite definir un nombre para el servicio. Éste es imprescindible a la hora de utilizar la aplicación con terminales reales ya que los móviles ofrecen de por sí diferentes servicios de todo tipo, L2CAP, SPP, HTTP… siendo imposible diferenciar un servicio ofrecido por el terminal o por el usuario sin este atributo. Este problema no ocurre con el simulador, ya que tan sólo existe el servicio que se ha creado anteriormente.

En este momento, la conexión ha sido creada, por lo que tan sólo hace falta utilizar los métodos descritos anteriormente para enviar y/o recibir datos.

2.1.1.6. Cliente SPP

Esta clase se encarga de descubrir dispositivos, mostrándolos en un `ChoiceGroup` y luego si el usuario desea, buscar los respectivos servicios y conectarse.

Antes de nada, se tiene que definir el UUID del cliente, que tiene que coincidir con el del servidor, ya que si no, no encontrará el servicio creado por éste. Acto seguido se sigue el mismo paso que en la clase anterior, es decir, recoger la configuración de bluetooth para así poder utilizar los servicios ofrecidos por éste. Una vez recogida y mediante el método `getDiscoveryAgent` del objeto `LocalDevice` se establece la visibilidad del móvil, devolviendo un objeto del tipo `DiscoveryAgent`.

Una vez llamada la función anterior, ya se puede empezar la búsqueda de dispositivos con `startInquiry`. Para poder obtener los móviles al alcance del terminal es necesario que la clase que utiliza el método anterior implemente `DiscoveryListener`, ya que el `startInquiry` inicializa los métodos `deviceDiscovered` e `inquiryCompleted`. Éstos son los encargados de notificar si un terminal es descubierto y el resultado de la búsqueda de los mismos.

El objeto móvil, dispone de diferentes métodos para poder obtener información acerca de él, siendo los más importantes;

- `getBluetoothAddress`: Obtiene la dirección física del terminal.
- `getRemoteDevice`: Devuelve el objeto móvil.
- `getFriendlyName`: Retorna el nombre del terminal.

Cada vez que el propio dispositivo encuentra otro, el método `deviceDiscovered` guarda el perfil del móvil encontrado en un vector llamado `devices` y un `ChoiceGroup` de nombre `listdevices`. Este último se muestra cuando el `inquiryCompleted` ha acabado correctamente; si no es así, se advierte al usuario del error que lo ha causado.

Mediante el metodo `searchServices` del objeto `DiscoveryAgent` se inicia la búsqueda de servicios. Del mismo modo que es necesario implementar dos métodos para recoger los dispositivos, en este caso ocurre exactamente lo mismo, se necesitan incluir los métodos `servicesDiscovered` y `serviceSearchCompleted` para recoger los servicios encontrados. El método que inicia la búsqueda, requiere de los parámetros:

- `Atributos`: Aquellos números que definen características del servidor, en este caso se utilizan 0x100, para definir el nombre, 0x101 para establecer el creador de la aplicación y 0x102 para poder reconocer el proveedor del servicio.
- `UUID`: Es el identificador único universal, 0x0003 para el caso del puerto serie RFCOMM.
- `RemoteDevice`: Con este parámetro seleccionamos en que dispositivo se quiere iniciar la búsqueda de servicios.
- `Listener`: Que clase implementa los métodos asociados a la búsqueda, en este caso es la clase `client_audio`.

Tal y como ocurre con la búsqueda de dispositivos, cuando se encuentra el servicio SPP buscado, éste se añade al vector `services`, para poder ser tratado posteriormente. Este servicio tan sólo se añade si coincide el nombre de éste con "`Server`", dado que es el único servicio que necesita buscar. Los objetos recogidos en el vector son del tipo ServiceRecord, con ciertos métodos que se resaltan a continuación:

- `getConnectionURL`: Obtiene la dirección del servicio.
- `getHostDevice`: Recoge el objeto móvil.
- `getAttributeValue`: Permite saber que atributos tiene el servicio, como por ejemplo el nombre.

Una vez encontrado el servicio, únicamente hace falta utilizar la clase `Connector` y los métodos comentados en el apartado 3.2.2.2 para establecer la comunicación.

En la figura (**Fig. 3.5**) se muestra un ejemplo de estas dos clases por separado y simuladas con la IDE del NetBeans. En este caso se puede ver el servidor iniciado con su dirección física, derecha, y el cliente en el momento que obtiene el servidor, izquierda.

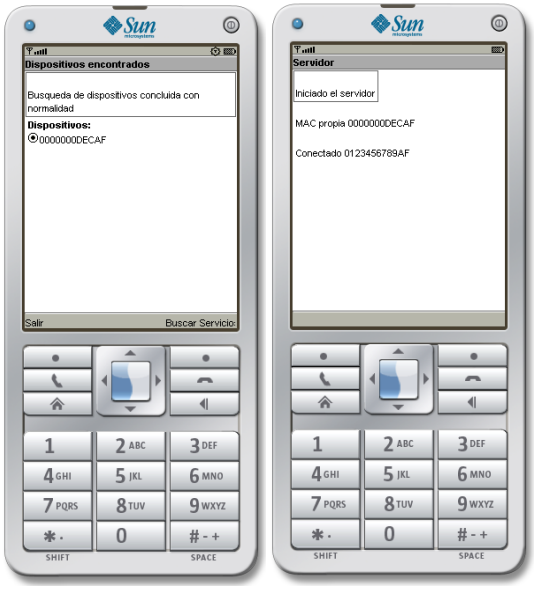

Fig. 3.5 Ejemplo de Cliente – Servidor.

2.1.1.7. Intercambio de audio

Debido a que un móvil tan sólo dispone de una aplicación, se tiene que combinar cliente y servidor en un mismo dispositivo. Ésto se debe a que un terminal debe implementar las dos clases, ya que un mismo terminal puede asumir tanto el rol de servidor como el de cliente.

La aplicación contiene los formularios necesarios para la interfaz grafica, ya que según las acciones del usuario, se debe mostrar una u otra información. Al pretender que la aplicación sea lo más sencilla posible, cuando un usuario desea establecer una comunicación, ya sea para intercambiar audio o vecinos, el otro extremo actualiza el formulario de su pantalla por el de la figura (**Fig. 3.6**). De esta forma, no es necesario que los dos o más usuarios inicien la aplicación al mismo tiempo. Además también dispone del `CommandListener` necesario para interactuar con el usuario.

Cabe resaltar que tanto el cliente como el servidor disponen de los métodos de búsqueda de dispositivos y servicios. El servidor requiere de éstos ya que los necesita para hacer de “puente” entre dos terminales, buscando y conectando al dispositivo que le exige dicho cliente.

Cuando se inicia la aplicación, automáticamente se inicia el servidor o `Server_audio` para poder recibir clientes en cualquier momento. Debido a que esta clase requiere el uso del terminal en ciertos momentos sin importar que estén haciendo las demás clases, se implementa con un hilo (*Thread*) para que se ejecute “paralelamente”. El Thread también es utilizado en el `audio_client`, ya que realiza la búsqueda de dispositivos y servicios. Seguidamente se muestra el formulario para la petición de audio o vecinos.

Fig.3.6 Formulario cuando se conecta un cliente.

Una vez el usuario visualiza el formulario (**Fig. 3.6**) puede empezar una conversación de audio con el dispositivo que ha seleccionado previamente, o bien, puede preguntarle a dicho terminal si dispone de vecinos.

En el primer caso, y una vez pulsado el comando "`Menú Audio`", el cliente obtiene el formulario (**Fig. 3.7**) donde puede reproducir, `play`, o grabar, `rec`, audio que más tarde se enviará al otro extremo de la comunicación. Debido al símil realizado con un "walkie-talkie", cuando se pulsa el `Command rec`, el usuario receptor de éste se le deshabilita el comando de grabar en el receptor, tan sólo puede reproducir el sonido. Una vez acabe de grabar el emisor, se restablece el `rec` en el receptor para poder enviar audio en sentido contrario o volver a escuchar el mensaje enviado anteriormente. A continuación se muestra un ejemplo de envío de audio mediante la aplicación. Se puede observar que el terminal de la izquierda es quien envía el audio y el de la derecha lo recibe.

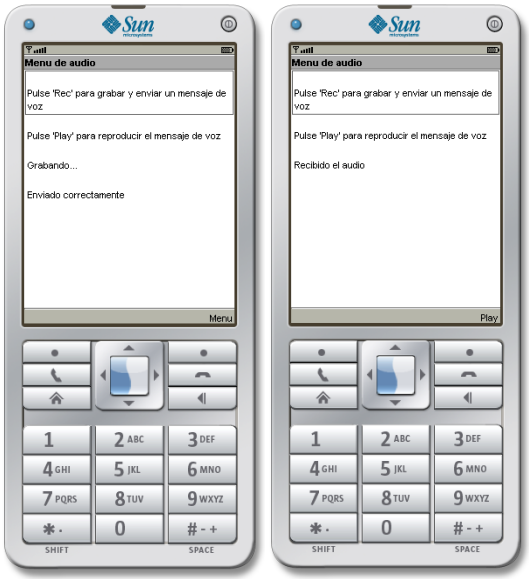

Fig. 3.7 Simulación de la función de audio.

Para diferenciar si un cliente desea establecer una comunicación audio u obtener los vecinos de éste, cuando el `Command` correspondiente es utilizado, se envía un `String`. En el caso de solicitar el audio, el `String` enviado es "`audio`", mientras que para los vecinos es "`router`". Cuando se recibe el `String` de "`audio`", el usuario que lo ha enviado facilita al otro extremo los parámetros de la figura (**Fig. 3.8**).

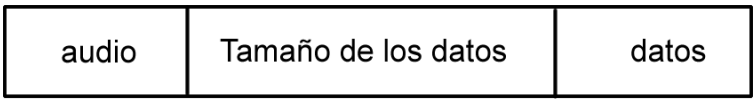

Fig. 3.8 Formato de petición de audio.

3.2.3. Red scatternet

En este apartado se explica como se realizan las conexiones mediante redes scatternet, más concretamente, como se han implementado estas conexiones entre servidor y cliente.

El problema de la comunicación cliente – servidor radica en el poco alcance de esta tecnología para la comunicación, aproximadamente 10m. (**Fig. 3.9**).
Debido a ello, este proyecto prevé el problema y actúa en consecuencia, permitiendo a los servidores conectarse con otro cliente. Es decir, si el terminal A está a rango del B, pero no del C, no se podrá comunicar con éste, pero gracias a la implementación del multi-salto.

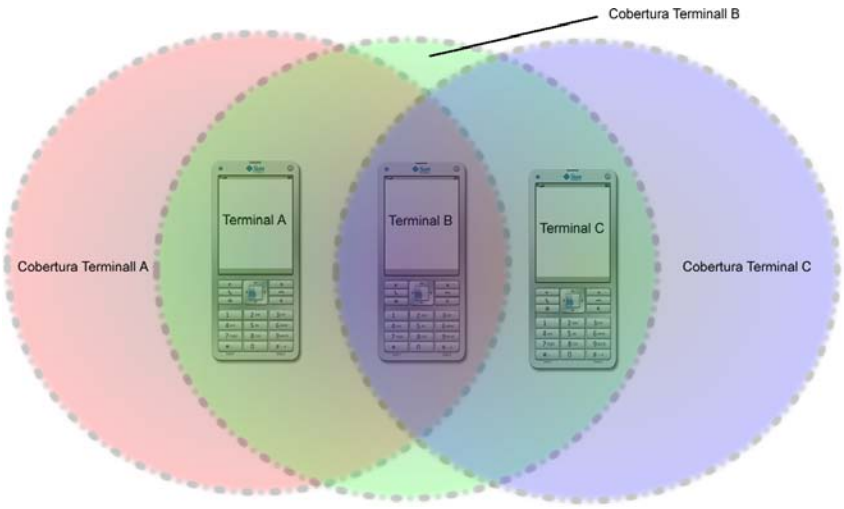

Fig. 3.9 Problemática del cliente – servidor.

Esta función está implementada en cada servidor, por lo que no tiene límite el número de saltos realizados con dicha aplicación.

Cuando al servidor le llega el `String` "router", se envía una serie de parámetros, detallados en la figura (**Fig. 3.10**)

router	Tamaño de vecinos	@ Vecinos

Fig. 3.10 Formato de petición de vecinos.

Cuando son recibidos todos los vecinos, al solicitante le aparece un formulario nuevo con la lista de vecinos, `listvecinos`, en formato `ChoiceGroup`, tal y como se muestra en la imagen (**Fig. 3.11**).

Fig. 3.11 `ChoiceGroup` de vecinos.

Una vez recibido y pulsado el `Command Selecciona`, el servidor inicia la búsqueda de servicios del dispositivo seleccionado. En el caso de encontrar el servicio, conecta directamente con él, obteniendo el formulario de la figura (**Fig. 3.6**). Una vez aquí, las conexiones se crean automáticamente y tan sólo hace falta iniciar el `Menu audio` u `Obtener vecinos`.

En la figura (**Fig. 3.12**) se ejemplifica el uso de esta función, siendo el terminal con dirección `0123456789AF` el extremo que pide los vecinos a `00006BD73E73` para alcanzar el dispositivo `0000000DECAF`. El móvil `00006BD73E73` ejerce de paso medio, y comunica los otros dos dispositivos. Debido a que el simulador no ofrece ninguna posibilidad de configuración respecto a los alcances de estos, esta prueba ha sido realizada de tal forma que todos comparten el medio físico. Aun así, la comunicación pasa a través de `00006BD73E73` para llegar al extremo opuesto. Se muestra a continuación un ejemplo gráfico de lo explicado anteriormente.

Fig. 3.12 Simulación de uso del multi-salto.

3.3. Fases de desarrollo

Debido a que se utiliza un lenguaje de programación orientado a objetos, J2ME, la clase principal se divide en diversas clases, cada una con una función, haciendo posible que la aplicación sea modular. Con esta característica se pretende que la aplicación pueda evolucionar en los servicios ofrecidos, ya que tan sólo hace falta implementar dicho servicio como una clase más junto a las ya existentes. En la figura (**Fig. 3.13**) se muestra el esquema de clases, detalladas en los siguientes apartados.

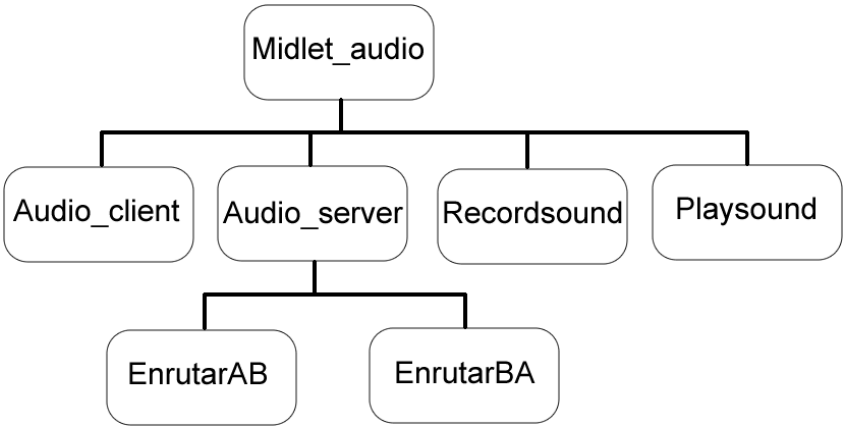

Fig. 3.13 Esquema de clases.

3.3.1. Midlet audio

Ésta es la clase principal, y de ella dependen todas las demás a excepción de `Vecino`, que se implementa aparte.

Extiende de `MIDlet`, por lo que implementa los métodos `startApp`, `pauseApp` y `destroyApp` correspondientes. El primero de ellos se llama cuando se inicia la aplicación, ejecutando a su vez a `iniciabluetooth`. `PauseApp` y `destroyApp` se utilizan cuando la aplicación está en pausa y cuando se sale del Midlet respectivamente.

`Iniciabluetooth` es el método encargado de recuperar la configuración bluetooth, configurar el teléfono como visible e iniciar el servidor `Audio_server`. Una vez iniciado, el usuario es libre de iniciar la búsqueda de dispositivos mediante el comando `Empezar` o bien salir de la aplicación.

Si se pulsa el `Command Empezar`, se inicia la clase `Audio_client`, realizando una búsqueda del medio para localizar dispositivos. Éstos son mostrados al usuario mediante un `ChoiceGroup`, como se ha visto en la sección 3.2.2.3. Tal y como ocurrió anteriormente, el usuario tiene la opción de buscar los servicios asociados a ese terminal o bien volver atrás. Si escoge la primera opción, la clase `Audio_client` se encargá de buscar los servicios mediante el método `searchServices`. Una vez encontrados, dispone de la opción conectar, que realiza la operación, de nombre similar, `conecta`. Esta función establece la conexión entre los dos terminales y devuelve un objeto `Vecino` cuando es llamada.

3.3.2. Audio client

`Audio client` implementa las funciones `Runnable` y `DiscoveryListener`, para poder ser ejecutada en paralelo y poder realizar búsquedas de dispositivos y/o servicios. Las clases que implementan la función `Runnable`, tienen como obligación implementar el método `run`, ya que cuando son llamadas se inicia el código dentro de dicho método.

En este caso, cuando se utiliza esta clase, `run` inicia la función `SearchDisp` para buscar dispositivos cercanos. Una vez llamada queda a elección del cliente buscar los servicios con el `Midlet_audio`. Para obtener el dispositivo seleccionado por el usuario, se ha creado la función `getchoice`, que devuelve un `String` con la opción escogida. En el caso de proseguir con la búsqueda, la clase anteriormente mencionada ejecuta el método `searchServices`, contenido dentro de ésta.

3.3.3. Audio server

Tal y como se ha comentado anteriormente, esta clase implementa las mismas funciones que `Audio_client`, es decir, `Runnable` y `DiscoveryListener`. Cuando el método `run,de la` propia la clase, se inicia el código contenido dentro de `creaService`. El nombre de ésta indica la función de la misma, que no es otra que crear el servicio del servidor. Una vez creado, el método devuelve un objeto del tipo `StreamConnectionNotifier`, necesario para aceptar los clientes con `acceptAndOpen`. Dicho objeto se recoge para ser introducido por parámetro en la función `conectaClientes`.

Como cabe de esperar, este método tiene la misión de conectar con los posibles clientes mediante un bucle, implementado con un `while`. Antes de nada, se inicia el valor de un `int`, de nombre `count`, para saber si es la primera vez que se conecta a un cliente o la segunda. En el caso de ser la segunda, significa que el usuario ha seleccionado la opción Obtener vecinos por lo que inicia la clase `EnrutarBA`, explicada en el apartado 3.3.6, pasando por parámetro el vecino obtenido en la segunda conexión. Si no es así, indica que es la primera vez que se conecta al servidor.

Una vez dentro de este bucle, se crea el objeto Vecino correspondiente al cliente conectado y entra en la espera para recibir un `String`. Éste permite diferenciar el servicio que desea el usuario, bien sea iniciar una conservación, obtener vecinos o informar que el cliente se ha desconectado, gracias al método de dicho objeto `compareTo`.

En el caso de una conversación de audio, el servidor entra en un bucle infinito, hasta que el cliente desconecte o bien salga de la aplicación, ejecutando la función `recibeaudio`. Dicho método se bloquea hasta que recibe el tamaño del array de datos, creando acto seguido un array con el tamaño recibido. Éste array representa los datos de audio y mediante `readFully` se recibe dicha

información. Una vez obtenido los datos, el usuario que los recibe ejecuta la función `delcomand` para deshabilitar la función de grabado del emisor, `rec`, y habilita la de reproducción del receptor, `play`, mediante `setcomand`.

Si es recibido el `String router`, la función `reciberouter` entra en acción, recibiendo por parámetro los flujos de entrada y salida del cliente conectado. La primera acción realizada por este método es la de enviar el numero de vecinos al otro extremo mediante `writeInt`. Mediante un `for`, de límite el tamaño de los vecinos, se envían las direcciones físicas de estos con `writeUTF`. Una vez acabado de enviar todos los vecinos, queda a la espera de recibir el vecino seleccionado con el mismo método utilizado para enviarlos. De esta forma, cuando recibe el vecino, de hecho recibe tan solo la dirección, busca en el vector `devices` dicho objeto con un `for` del mismo tamaño que el anterior e inicia `searchServices`, con la diferencia que la propia clase recibe los vecinos. Si es encontrado el vecino seleccionado conecta directamente con él, creando así el "puente" entre los dos terminales, iniciando la clase `EnrutarAB`.

3.3.4. Recordsound

Esta clase es la encargada de registrar el audio recibido por el altavoz del móvil, para luego enviarlo al usuario conectado. Por tanto, dispone de dos métodos bien diferenciados; `rec` y `send`.

El primero de ellos, graba el audio, gracias a la API `javax.microedition.media.control.RecordControl` que permite dicho uso. Antes de iniciar la grabación, se tiene que iniciar el método `createPlayer` de la clase abstracta `Manger`, recibiendo como parámetro el código `capture://audio?encoding=pcm`. Ésto define que tipo de audio se va a grabar, estando codificado en `pcm` [10], y de donde procede el audio, en este caso capturado. El objeto devuelto por dicha clase es del tipo `Player`, específico de la API mencionada.

Acto seguido, se crea el objeto `RecordControl`, que no es más que la ejecución del método `getControl` del objeto `Player`. Éste permite definir donde guardar los datos una vez estén grabados. En este caso, son grabados en un buffer tipo `ByteArrayOutputStream`, donde más tarde serán copiados a un arary de bytes. Seguidamente, para poder relacionar dicho objeto con el `Player`, se utiliza el método `setRecordStream`, donde se pasa por parámetro el buffer. Para iniciar la grabación, se utiliza el método `startRecord` del objeto antes mencionado junto con `start` del `Player`.

Debido a las limitaciones de memoria de los terminales, no es posible grabar audio de forma indefinida, por tal motivo, tan sólo se pueden grabar 4 segundos de audio. Esta función es implementada con el método `sleep` del propio `Thread` en ejecución. Una vez haya pasado ese tiempo o haya sido cancelado, los objetos anteriormente abiertos tienen que ser cerrados, con las funciones `stopRecord` y `stop`. Para poder grabar el audio al buffer, es necesario el uso

del método `commit` del objeto `RecordControl`, siempre teniendo en cuenta que previamente se tiene que haber cerrado los objetos abiertos con anterioridad. Llegado este momento, tan solo falta copiar la grabación al array de bytes creado anteriormente mediante la función `toByteArray`.

Como cabe de esperar, el método `send` es utilizado para enviar la grabación en formato array de bytes al otro extremo de la conexión. Relacionado con la figura (**Fig. 3.8**), envía el mensaje de audio seguido del tamaño de los datos junto con éstos.

3.3.5. Playsound

Tal y como hay una clase para grabar audio, también hay otra para poder reproducirlo. A diferencia de `RecordSound`, esta clase es más fácil de implementar ya que no necesita tantos objetos y funciones para reproducir audio, además de utilizar `PlayerListener`, muy parecido a `CommandListener`.

Previamente se tiene que haber recibido el array de bytes gracias al método `recogearray` de la presente clase. Una vez recogida, se inicia la reproducción con la función `play`.

Esta función crea un objeto del tipo `ByteArrayOutputStream`, indicando como parámetro el array previamente recogido. Una vez creado, se inicia el método `createPlayer` del objeto `Manager`, igual que en la clase anterior, con la diferencia de pasar por parámetro el objeto que contiene el array y el formato del audio, en este caso `audio/x-wav`. Tan sólo queda activar la escucha de dicha clase con `addPlayerListener` e iniciar la reproducción mediante `start`.

`addPlayerListener` implementa el método `playerUpdate`, el cual indica el estado de la reproducción. Dependiendo del estado, se recibe una notificación. En este caso, cuando llega un evento al método y es del tipo `endOfMedia`, indican que la reproducción ha finalizado, por lo que se vuelve a añadir el comando `rec` al emisor.

3.3.6. EnrutarAB

Esta clase, y la tratada en el siguiente apartado, tienen la misma función, pero de forma invertida. Es decir, esta clase trata de escuchar los mensajes enviados por el usuario que ha pulsado el comando `Obtener vecinos`, A, y se los envía a B, que es el usuario que recibe dicho comando.

Cunado se crea, recoge las diferentes conexiones y las guarda en dos objetos tipo `Vecino`, para facilitar su uso. Debido a que espera de forma indefinida la llegada de datos, tiene que heredar de la clase `Thread`, para poder utilizar dicha clase de forma paralela gracias al método `run`.

Su función es la de emular el método `recibeaudio`, pero con la característica de que funciona en un sólo sentido de la comunicación. Los datos enviados por A son recogidos en B y luego enviados al vecino de este último, C. Estos datos no son más que los reflejados en la figura (**Fig. 3.8**). Una vez recogido el `String`, entra en un bucle para poder recoger el audio de forma continuada.

3.3.7. EnrutarBA

Como ya había sido avanzado anteriormente, esta clase tiene como propósito recoger y reenviar los datos. Siguiendo con el ejemplo anterior, capta los datos enviados de C a B para luego enviarlos hacia A.

Respecto a la clase anterior, sólo existe la diferencia de los vecinos que utiliza éste para la comunicación, aunque recoge los mismos datos que la clase anterior, por lo que queda explicado en el anterior apartado.

3.3.8. Vecino

Esta clase ha sido creada para facilitar el intercambio de flujos entre cliente y servidor, ya que al utilizar varias conexiones simultáneas el número de flujos de éstas se incrementa en un factor de 2, siendo una entrada y salida por cada flujo.

La clase dispone de dos constructores, uno vacío y otro con un `String` como parámetro. Éste indica la dirección física de los flujos establecidos

Dispone de los siguientes métodos:

- `setMac_address`: Da valor al `String` anterior si no ha sido iniciada en el constructor.
- `getMac_address`: Obtiene la dirección física del flujo.
- `setConn`: Establece el valor del objeto `StreamConnection`.
- `getConn`: Devuelve el `StreamConnection`.
- `setIn`: Inicia el valor del `DataInputStream`.
- `getIn`: Recoge el flujo de salida.
- `setOut`: Define el `DataOutputStream`.
- `getOut`: Obtiene el valor del flujo de entrada.
- `cerrarConexiones`: Cierra las conexiones del propio objeto.

3.4. Muestra de funcionamiento sobre terminales reales

En este apartado se pasa a detallar con imágenes el funcionamiento de la aplicación. Debido a que dicha aplicación se basa en reproducir audio, se debe hacer especial hincapié en los datos mostrados por las pantallas.

Para comprobar el funcionamiento, se han realizado dos pruebas, una conversación entre dos terminales, y otra con tres, tal y como se presenta en la figura (**Fig. 3.9**).

3.4.1. Descripción de los dispositivos

Los tres móviles escogidos para la comprobación de la aplicación son de la marca Nokia, ya que actualmente, es una de las compañías líderes del sector.

Debido al número limitado de SIMs disponibles, se a buscado móviles que soporten la transmisión de datos por bluetooth sin tener conexión de red.

2.1.1.8. Nokia 6120 Classic

Las características más remarcables de este móvil son:

- Interfaz de usuario serie 60.
- Soporta los formatos WMA, MP3, M4A y eAAC+ de audio.
- Admite los formatos de voz: NB-AMR, AMR y FR.
- Memoria para usuario de 35 Mb.
- Bluetooth 2.0 con perfil de manos libres, acceso a SIM y A2DP.
- Permite la transmisión de datos *offline* (sin línea).
- Comandos y grabadora de voz.
- Dispone de MIDP 2.0

Fig. 3.14 Nokia 6120 Classic.

2.1.1.9. Nokia 6110 Navigator

Este móvil dispone de unas características similares al anterior:

- Interfaz de serie 60.
- Soporta los formatos MP3,MP4, ACC+,WMA de audio.
- Admite codecs de voz para NB-AMR, AMR, FR y EFR.
- Memoria disponible para el usuario de 40Mb.
- Bluetooth 2.0 + EDR, con transmisión de datos sin línea.
- MIDP 2.0

Fig. 3.15 Nokia 6110 Navigator.

2.1.1.10. Nokia N73

A continuación se resaltan las características principales:

Interfaz de usuario serie 60.
Permite los codecs eAAC+, AAC, ACC+WMA y MP3 de audio.
Codecs de voz admitidos: FR, EFR, WCDMA y GSM AMR.
Memoria máxima disponible para el usuario de 42 Mb.
Bluetooth 2.0 con transmisión de datos *offline*.
Kit de creación de aplicaciones Java y C++.
MIDP 2.0, CLDC 1.1, 3D API y PIM API.

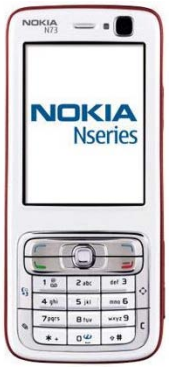

Fig. 3.16 Nokia N73.

3.4.2. Ejemplo de conversación sin scatternet

Para realizar la comprobación de la aplicación mediante dos terminales que compartan el medio físico, se ha utilizado el Nokia 6120 Classic y el Nokia 6110 Navigator.

En este caso el móvil 6120 es quien inicia la aplicación para conectarse al otro terminal. Seguidamente se detallan las direcciones físicas de los dispositivos para una mayor comodidad.

Nokia 6120 Classic: `0021AA179AE2`
Nokia 6110 Navigator: `001CD6653C67`

En la figura (**Fig. 3.17**) se muestra el formulario inicial de ambos dispositivos móviles, dejando constancia de la dirección física.

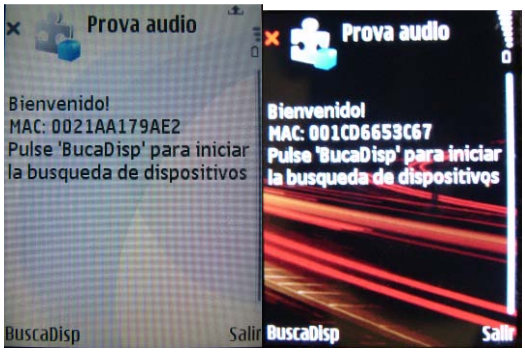

Fig. 3.17 `Form` principal.

El resultado de buscar dispositivos es la figura (**Fig. 3.18**). Una vez buscado los servicios, se ofrece la posibilidad de conectarse. Si un terminal no dispone de la aplicación, la conexión, como era de esperar, no será posible.

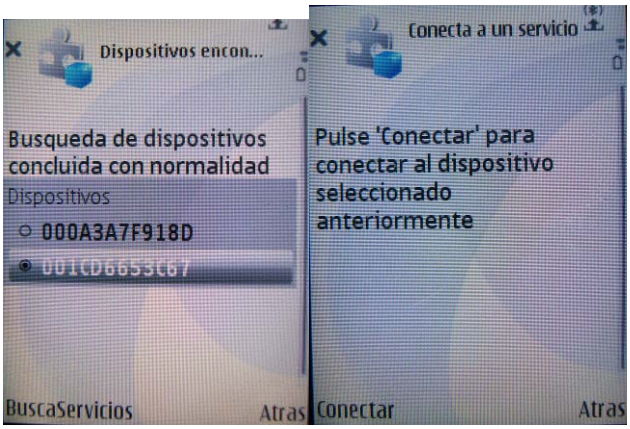

Fig. 3.18 Búsqueda de dispositivos y servicios.

Cuando se conecta al otro terminal, éste muestra un aviso al usuario, advirtiendo que se está conectando al dispositivo mostrado. Cuando éste acaba de conectar, muestra el formulario derecho de la figura (**Fig. 3.19**)

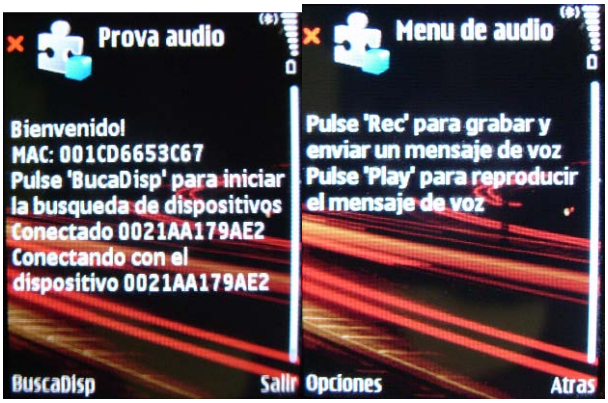

Fig. 3.19 Fases de conexión a un dispositivo remoto.

Una vez se ha obtenido el formulario con los comandos `Menú Audio` y `Obtener Vecinos`, la aplicación llega a su último formulario, si el cliente no desconecta. En la figura (**Fig. 3.20**) se muestra el envío de una grabación de voz.

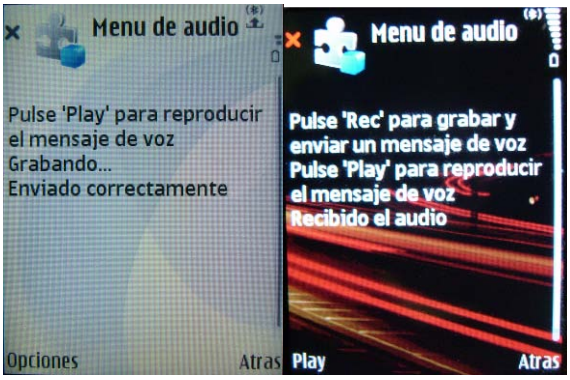

Fig. 3.20 Envío y recepción de audio.

Se puede observar que el terminal de la derecha dispone únicamente del comando play para reproducir el audio procedente del otro extremo del flujo.

3.4.3. Ejemplo de conversación con scatternet

En este apartado se ha probado la comunicación de dos terminales pasando por un tercero a modo de puente.

Las direcciones físicas de cada dispositivo son mostradas a continuación:

Nokia 6120 Classic: `0021AA179AE2`
Nokia 6110 Navigator: `001CD6653C67`
Nokia N73: `0021082ED9F5`

Concretamente en esta prueba, se ha utilizado el teléfono 0021AA179AE2 como usuario que pide la conexión puente a `0021082ED9F5` para llegar al dispositivo `001CD6653C67`.

Acto seguido se muestra en la figura (**Fig. 3.21**) las pantallas iniciales de los terminales.

Fig. 3.21 Formularios iniciales de los tres móviles.

El orden de las pantallas en la figura (**Fig. 3.22**) indica el orden de conexión, empezando por la izquierda hasta llegar a la derecha.

Fig. 3.22 Búsqueda de dispositivos del `0021AA179AE2`.

Tal y como ocurrió en el apartado anterior, cuando el `0021AA179AE2` pulsa el comando de conectar, el otro dispositivo advierte al usuario.

Fig. 3.23 `0021AA179AE2` y `0021082ED9F5` conectados.

En este caso se escoge la opción Obtener vecinos, por lo que el `0021AA179AE2` muestra los vecinos que tiene `0021082ED9F5` al alcance en la figura (**Fig. 3.24**).

Fig. 3.24 Obtención de vecinos de `0021AA179AE2`.

Cuando el usuario opta por seleccionar el dispositivo, vuelve a ocurrir lo mismo que si estuviera pulsando el boton de conectar en la demostracion anterior.

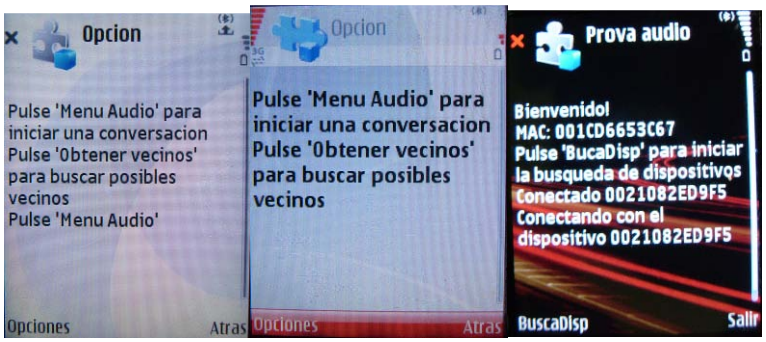

Fig. 3.25 Conectando con el dispositivo `001CD6653C67`.

Una vez la conexión ha sido establecida, se muestra en la figura (**Fig. 3.26**) el envío y recepción de audio por parte de los dos terminales.

Fig. 3.26 Envío y recepción de audio a través de un tercero.

CAPÍTULO 4 CONCLUSIONES

Este capítulo concluye con las conclusiones recogidas a lo largo del desarrollo de la aplicación, como también las líneas futuras posibles y un pequeño estudio sobre el impacto de dicha aplicación en el medio ambiente.

4.1. Conclusiones

En éste proyecto se ha realizado una aplicación para transmitir mensajes de voz, como un “walkie-talkie”, utilizando redes bluetooth multi-salto ad-hoc, resultando ser una aplicación gratuita y funcional.

Se ha elegido dichas redes por ser una de las tecnologías más extendidas en estos dispositivos, además de tener un consumo de batería reducido, siendo un problema para WLAN. Aún siendo la más extendida, su implementación en los terminales actuales no es la idónea, por lo que no es compatible al 100% en todos los teléfonos móviles. Se ha optado por el lenguaje de programación Java, ya que garantiza la compatibilidad en los diferentes sistemas operativos y proporciona, tanto fiabilidad como estabilidad.

La aplicación se ha desarrollado con el software NetBeans 6.7, que a lo largo del desarrollo de la aplicación ha demostrado ser una IDE excelente, tanto por su simulador integrado como por su facilidad de uso. J2ME se ha consagrado como lenguaje de programación en los terminales, gracias sobre todo, a la compatibilidad mostrada junto con la orientación a objetos heredada de Java.

Durante el desarrollo del TFC se han encontrado dificultades a la hora de testearlo en móviles reales. Debido a la configuración implementada en todos ellos, dichos terminales no permiten el intercambio de archivos *.jar*, que son los programas compilados en J2ME listos para la instalación, por motivos de seguridad. Este problema rompe con la idea de distribución de terminal a terminal, obligando a descargarlo desde un ordenador con cable o bluetooth. También se han encontrado problemas a la hora de ejecutar la aplicación, ya que no todos los móviles disponen de la memoria suficiente para la aplicación. Este problema se añade a la dificultad de configuración de determinados terminales para la utilización de Java con las funciones del terminal, ya que no disponen de un menú de configuración completo, como es el caso de los terminales fabricados por LG.

Aun así, la aplicación que se ha desarrollado es completamente funcional y operativa salvo en determinados casos ya comentados, por lo que los objetivos globales de este TFC han sido cumplidos.

4.2. Líneas futuras

Debido a la limitación de memoria de los terminales junto con la falta de depuración del código, queda para revisión una mejora del código para poder comunicarse con más de una persona a la vez. Es decir, emular un servicio *multicast* donde las personas inscritas en un servicio, por tanto compartiendo el medio, puedan recibir y enviar mensajes de voz.

La interfaz gráfica también es un aspecto a mejorar, ya que se ha diseñado de forma simple pero funcional. A la vista quedan las posibilidades de efectos visuales, implementación de sonidos, avisos…

Tal vez la corta cobertura de bluetooth es un punto a mejorar en versiones posteriores implementando la tecnología Wifi, aun a costa de acortar la vida del terminal. Aunque este apunte queda suspendido hasta que se aporte una solución al consumo de Wifi seguramente por la fabricación de baterías más duraderas.

En la sociedad actual, la protección de los datos es muy importante, siendo crítica en temas relacionados con vulnerabilidades de bluetooth. Por tal motivo se espera una mejora efectiva en las siguientes versiones del mismo. En este aspecto, la aplicación que se ha realizado flaquea, ya que las comunicaciones no han sido encriptadas, como tampoco se requiere autenticación para establecerlas.

Como apunte final, hubiera podido ser habilitado el envió de archivos .jar, ya que con esta herramienta la aplicación hubiera tenido un carácter mas social al poder compartir aplicación sin tener que descargarla.

4.3. Estudio de ambientalización

Debido a que este TFC consta de realizar una aplicación para móviles, tan sólo se pueden encontrar efectos de contaminación en el área de radiaciones electromagnéticas.

Bluetooth opera con un bajo radio de acción, tan solo 10 metros, por lo que la potencia con la que emite es muy baja, entre los 0,25mW a 0,5mW. Estas potencias son prácticamente inocuas para el ser humano, además de ser de las más bajas comparadas con las efectuadas por: Wifi, TV, radio, microondas… Por lo tanto, el impacto medioambiental que puede producir las radiaciones realizadas mediante esta tecnología inalámbrica se podría considerar despreciable.

REFERENCIAS

[1] http://es.wikipedia.org/wiki/C%2B%2B

[2] http://en.wikipedia.org/wiki/Wireless_LAN

[3] http://thesmartgenius.wordpress.com/2008/05/11/bluejacking-bluesnarfing-bluebugging-bluetoothing/

[4] http://thesmartgenius.wordpress.com/2008/05/11/bluejacking-bluesnarfing-bluebugging-bluetoothing/

[5] http://es.wikipedia.org/wiki/Java_Database_Connectivity

[6] http://en.wikipedia.org/wiki/WWW

[7] http://www.scribd.com/doc/6260482/Historia-de-Java

[8] http://es.wikipedia.org/wiki/Common_Development_and_Distribution_License

[9] http://en.wikipedia.org/wiki/Universally_Unique_Identifier

[10] http://en.wikipedia.org/wiki/PCM

Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TITULO DEL TFC: Aplicación "pulsar para hablar" para dispositivos móviles con compatibilidad JME y bluetooth

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Telemática

AUTOR: Jose Luis Puig Guerra

DIRECTOR: Juan Hernández Serrano

FECHA: 17 de Julio de 2009

A continuación se muestran los Javadocs pertinentes a las clases `Midlet_audio` y `Vecinos`.

1.1. Annexo 1: Javadoc clase Midlet audio

JAVADOC

Class Midlet_audio

```
java.lang.Object
  └ javax.microedition.midlet.MIDlet
      └ TFC.Midlet_audio
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

```
public class Midlet_audio
extends javax.microedition.midlet.MIDlet
implements javax.microedition.lcdui.CommandListener
```

Esta clase es la principal, ya que contiene el servidor, cliente y demás funciones. Se encarga de iniciar los servicos, como tambien de buscarlos y conectarlos.

Constructor Summary

Midlet_audio()	Constructor deMidlet_audio.

Method Summary

void	**buscaservice**(javax.bluetooth.RemoteDevice neigh) Realiza la busqueda de servicios.
void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Define las funciones que realizan los comandos.
Vecino	**conecta**(java.lang.String mac) Conecta con el servicio seleccionado.
void	**delcomand**(javax.microedition.lcdui.Command com) Deshabilita el comando c.
void	**destroyApp**(boolean unconditional) Es llamado cuando se cierra la aplicación.
javax.microedition.lcdui.Display	**getDisplay**() Obtiene el display actual.
javax.microedition.lcdui.Form	**getinciaformdevices**() Obtiene el formulario iniciaformdevices.
javax.microedition.lcdui.Form	**getiniciaaudio**()

	Obtiene el formulario iniciaaudio.
javax.microedition.lcdui.Form	**getiniciaconecta**() Obtiene el formulario conecta.
javax.microedition.lcdui.Form	**getiniciaform**() Obtiene el formulario iniciaform.
javax.microedition.lcdui.Form	**getiniciamenu**() Obtiene el formulario iniciamenu.
javax.microedition.lcdui.Form	**getiniciamenuvecinos**() Obtiene el formulario iniciamenuvecinos.
javax.microedition.lcdui.Form	**getiniciaservice**() Obtiene el formulario iniciaservice.
javax.microedition.lcdui.Form	**inciaformdevices**() Inicia el formulario iniciaformdevices.
javax.microedition.lcdui.Form	**iniciaaudio**() Inicia el formulario iniciaaudio.
javax.microedition.lcdui.Form	**iniciaconecta**() Inicia el formulario iniciaconecta.
javax.microedition.lcdui.Form	**iniciaform**() Inicia el formulario iniciaform.
javax.microedition.lcdui.Form	**iniciamenu**() Inicia el formulario iniciamenu.
javax.microedition.lcdui.Form	**iniciamenuvecinos**() Inicia el formulario iniciamenuvecinos.
javax.microedition.lcdui.Form	**iniciaservice**() Inicia el formulario iniciaservice.
void	**pauseApp**() Es llamado cuando la aplicación está en pause.
java.util.Vector	**recibevecino**() Obtiene los vecinos disponibles, devuelve un vector.
java.lang.String	**seleccion**() Devuelve el terminal que ha elegido el usuario.
java.lang.String	**selecvecino**() Devuelve el terminal vecino que ha elegido el usuario.
void	**setcomand**(javax.microedition.lcdui.Command com) Habilita el comando c.
void	**setdisplay**(javax.microedition.lcdui.Displayable nextdisp, javax.microedition.lcdui.Alert alert) Impone el display nextdisp.
void	**startApp**() Es llamado cuando se inicia la aplicación.

Methods inherited from class javax.microedition.midlet.MIDlet
getAppProperty, notifyDestroyed, notifyPaused, resumeRequest

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Midlet_audio

```
public Midlet_audio()

public java.lang.String seleccion()
```

---

buscaservice

```
public void buscaservice(javax.bluetooth.RemoteDevice neigh)
```

---

conecta

```
public Vecino conecta(java.lang.String mac)
```

---

recibevecino

```
public java.util.Vector recibevecino()
```

---

selecvecino

```
public java.lang.String selecvecino()
```

Method Detail

startApp

```
public void startApp()
```

Specified by:

`startApp` in class `javax.microedition.midlet.MIDlet`

Es llamado cuando se inicia la aplicación, llama al método iniciabluetooth.

---

pauseApp

```
public void pauseApp()
```

Specified by:
`pauseApp` in class `javax.microedition.midlet.MIDlet`
No ha sido implementado.

---

destroyApp

```
public void destroyApp(boolean unconditional)
```

Specified by:
`destroyApp` in class `javax.microedition.midlet.MIDlet`
Se utiliza cuando se salde del midlet, borra todos los objetos creados.

---

iniciaform

```
public javax.microedition.lcdui.Form iniciaform()
```

Inicia el formulario de bienvenida.

---

getiniciaform

```
public javax.microedition.lcdui.Form getiniciaform()
```

Recoge el formulario principal.

---

inciaformdevices

```
public javax.microedition.lcdui.Form inciaformdevices()
```

Inicia el formulario donde se representan los vecinos obtenidos.

---

getinciaformdevices

```
public javax.microedition.lcdui.Form getinciaformdevices()
```

Recoge el formulario donde se representan los vecinos obtenidos.

---

iniciaconecta

```
public javax.microedition.lcdui.Form iniciaconecta()
```

Inicia el formulario donde se escoge la opción de conectar.

---

getiniciaconecta

```
public javax.microedition.lcdui.Form getiniciaconecta()
```

Obtiene el formulario que contiene el comando conectar.

---

iniciamenu

```
public javax.microedition.lcdui.Form iniciamenu()
```

Inicia el formulario con las opciones de Menu audio y Obtener vecinos.

---

getiniciamenu

```
public javax.microedition.lcdui.Form getiniciamenu()
```
Recoge el formulario que implementa los comandos menuaudio y obtenervecinos.

---

iniciaservice

```
public javax.microedition.lcdui.Form iniciaservice()
```
Inicia el formulario donde se reflejan los servicios obtenidos. Para el usuario es transparente, es decir, no llega a verlos.

---

getiniciaservice

```
public javax.microedition.lcdui.Form getiniciaservice()
```
Obtiene el valor del formulario que contiene los servicios.

---

iniciaaudio

```
public javax.microedition.lcdui.Form iniciaaudio()
```
Inicia el formulario que tiene los menus de rec y play.

---

getiniciaaudio

```
public javax.microedition.lcdui.Form getiniciaaudio()
```
Recoge el valor del formulario con los comandos rec y play.

---

iniciamenuvecinos

```
public javax.microedition.lcdui.Form iniciamenuvecinos()
```
Inicia el formulario con los vecinos obtenidos.

---

getiniciamenuvecinos

```
public javax.microedition.lcdui.Form getiniciamenuvecinos()
```
Devuelve el formulario con el ChoiceGroup de los vecinos.

---

setcomand

```
public void setcomand(javax.microedition.lcdui.Command com)
```
Inserta un comando el formulario.

---

delcomand

```
public void delcomand(javax.microedition.lcdui.Command com)
```

Elimina un comando del formulario.

---

getDisplay

```
public javax.microedition.lcdui.Display getDisplay()
```

Obtiene el objeto display actual.

---

setdisplay

```
public void setdisplay(javax.microedition.lcdui.Displayable nextdisp,
                       javax.microedition.lcdui.Alert alert)
```

Inserta el display o alerta pasado por parámetro.

---

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Specified by:

`commandAction` in interface `javax.microedition.lcdui.CommandListener`

Permite la interacción con el usuario mediante comandos.

---

seleccion

```
public java.lang.String seleccion()
```

Obtiene el vecino seleccionado por el usuario remoto.

---

buscaservice

```
public void buscaservice(javax.bluetooth.RemoteDevice neigh)
```

Inicia la función de buscar servicios en el cliente.

---

conecta

```
public Vecino conecta(java.lang.String mac)
```

Ejecuta el método para conectar con los clientes. Devuelve un objeto tipo Vecino.

---

recibevecino

```
public java.util.Vector recibevecino()
```

Cuando es ejecutada, devuelve el numero de vecinos obtenidos gracias al otro extremo de la conexión.

---

selecvecino

```
public java.lang.String selecvecino()
```

Devuelve el vecino que se ha seleccionado para después enviarlo por el flujo de salida.

1.2. *Annexo 2:* Javadoc clase Vecino

JAVADOC

Class Vecino

java.lang.Object
  └ **TFC.Vecino**

---

public class **Vecino**
extends java.lang.Object

Esta clase pretende facilitar la utilización de los flujos, tanto de salida como de entrada. Además dispone de métodos específicos para obtener las direcciones físicas y los diferentes objetos de la comunicación.

Constructor Summary

Vecino()	Constructor vacio.
Vecino(java.lang.String mac_address)	Constructor que inicia la variable de la dirección física.

Method Summary

protected void	**cerrarConexiones**() Cierra las conexiones creadas por el vecino.
javax.microedition.io.Connection	**getConn**() Obtiene el objeto DataOutputStream.
java.io.DataInputStream	**getIn**() Obtiene el objeto DataInputStream.
java.lang.String	**getMac_address**() Obtiene la dirección física.
java.io.DataOutputStream	**getOut**() Obtiene el objeto StreamConnection.
void	**setConn**(javax.microedition.io.StreamConnection conn) Inicia el objeto StreamConnection.
void	**setIn**(java.io.DataInputStream in) Inicia el objeto DataInputStream.
void	**setMac_address**(java.lang.String mac) Inicia el String de la dirección física.

void	**setOut**(java.io.DataOutputStream out) Inicia el objeto DataOutputStream.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Vecino

```
public Vecino(java.lang.String mac_address)
```

Vecino

```
public Vecino()
```

Method Detail

setMac_address

```
public void setMac_address(java.lang.String mac)
```
Inicia la variable que representa la dirección física.

---

getMac_address

```
public java.lang.String getMac_address()
```
Obtiene el valor del String mac_adress.

---

setOut

```
public void setOut(java.io.DataOutputStream out)
```
Inicia el objeto DataOutputStream necesario para escribir datos.

---

setConn

```
public void setConn(javax.microedition.io.StreamConnection conn)
```
Inicia el StreamConnection, necesario para aceptar clientes.

---

getConn

```
public javax.microedition.io.Connection getConn()
```

Obtiene el StreamConnection, con él se puede obtener el dispositivo remoto.

---

setIn

```
public void setIn(java.io.DataInputStream in)
```
Inicia el valor del DataInputStream, indispensable para leer datos.

---

getIn

```
public java.io.DataInputStream getIn()
```
Obtiene el valor del flujo de entrada.

---

getOut

```
public java.io.DataOutputStream getOut()
```
Obtiene el valor del flujo de salida.

---

cerrarConexiones

```
protected void cerrarConexiones()
```
Cierra todas las conexiones creadas por el vecino.